

AD-A198 890

JMC FILE CUE

2

AVF Control Number: AVF-VSR-90502/33

Ada* Compiler
VALIDATION SUMMARY REPORT:
Certificate Number:#871126N1.09001
University of York
University of York Ada Compiler, Release 3
Sun 3/50

Completion of On-Site Testing:
26th November 1987

Prepared By:
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
United Kingdom

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

SDTIC
ELECTE
SEP 01 1988
3 E D

*Ada is a registered trademark of the United States Government
(Ada Joint Program Office).

This document has been approved
for public release and sale in
distribution is unlimited.

88 8 31 023

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	12. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: University of York, University of York Ada Compiler, Release 3, Sun 3/50 (Host and Target).		5. TYPE OF REPORT & PERIOD COVERED 26 Nov 1987 to 26 Nov 1988
7. AUTHOR(s) National Computing Centre Limited, Manchester, United Kingdom.		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION AND ADDRESS National Computing Centre Limited, Manchester, United Kingdom.		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) National Computing Centre Limited, Manchester, United Kingdom.		12. REPORT DATE 26 November 1987
		13. NUMBER OF PAGES 39 p.
		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report) UNCLASSIFIED		
18. SUPPLEMENTARY NOTES		
19. KEYWORDS (Continue on reverse side if necessary and identify by block number) Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD- 1815A, Ada Joint Program Office, AJPO		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) University of York Ada Compiler, Release 3, University of York, National Computing Centre Limited, Sun 3/50 under UNIX (SunOS 3.2) (Host and Target), ACVC 1.9.		

Ada* Compiler Validation Summary Report:

Compiler Name: University of York Ada Compiler, Release 3

Certificate Number: #871126N1.09001

Host:

Sun 3/50
UNIX (SunOS 3.2)

Target:

Sun 3/50
UNIX (SunOS 3.2)

Testing completed 26 November 1987 Using ACVC 1.9

This report has been reviewed and is approved.

A.E.J. Pink

The National Computing Centre Ltd
Jane Pink
Oxford Road
Manchester M1M 7ED
United Kingdom

John F. Kramer

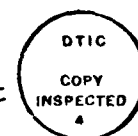
Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

*Ada is a registered trademark of the United States Government
(Ada Joint Program Office).



EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the University of York Ada Compiler, Release 3, using Version 1.9 of the Ada* Compiler Validation Capability (ACVC). The University of York Ada Compiler is hosted on a Sun 3/50 operating under UNIX (SunOS 3.2).

On-site testing was performed 25th November 1987 through 26th November 1987 at University of York under the direction of the NCC (AVF), according to Ada Validation Organisation (AVO) policies and procedures. At the time of testing, version 1.9 of the ACVC comprised 3122 tests of which 25 had been withdrawn. Of the remaining tests, 399 were determined to be inapplicable to this implementation. Results for processed Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 399 of the processed tests determined to be inapplicable. The remaining 2698 tests were passed. The results of validation are summarized in the following table:

RESULT	CHAPTER														TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14		
Passed	175	449	460	245	166	98	139	326	129	36	232	3	240	2698	
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Inapplicable	29	124	215	3	0	0	4	1	8	0	2	0	13	399	
Withdrawn	2	13	2	0	0	1	2	0	0	0	2	1	2	25	
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122	

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT	1-2
1.3	REFERENCES	1-3
1.4	DEFINITION OF TERMS	1-3
1.5	ACVC TEST CLASSES	1-4
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED	2-1
2.2	IMPLEMENTATION CHARACTERISTICS	2-2
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER	3-2
3.4	WITHDRAWN TESTS	3-2
3.5	INAPPLICABLE TESTS	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS	3-5
3.7	ADDITIONAL TESTING INFORMATION	3-5
3.7.1	Prevalidation	3-5
3.7.2	Test Method	3-6
3.7.3	Test Site	3-6
APPENDIX A	CONFORMANCE STATEMENT	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL STD 1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:-

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any unsupported language constructs required by the Ada Standard
- . To determine that the implementation-dependent behaviour is allowed by the Ada Standard

Testing of this compiler was conducted by NCC under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 25th November 1987 through 26th November 1987 at University of York.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:-

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:-

The National Computing Centre Ltd
Oxford Road
Manchester M1 7ED
United Kingdom

INTRODUCTION

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:-

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language,
ANSI/MIL-STD-1815A, February 1983.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint
Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide,
SofTech, Inc., December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These commentaries are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for establishing procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

INTRODUCTION

Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Language Maintenance Panel	The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and possible changes to the ANSI/MIL-STD for Ada.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	An Ada program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

INTRODUCTION

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

INTRODUCTION

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonable portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain for an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:-

Compiler: University of York Ada Compiler, Release 3

ACVC Version: 1.9

Certificate Number: #871126N1.09001

Host Computer:

Machine: Sun 3/50

Operating System: UNIX (SunOS 3.2)

Memory Size: 4Mb

Target Computer:

Machine: same as host

Operating System: same as host

Memory Size: same as host

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- . Capacities.

The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- . Universal integer calculations

An implementation is allowed to reject universal integer calculations having values that exceed `SYSTEM.MAX_INT`. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B).

- . Predefined types.

This implementation supports the additional predefined types `SHORT_INTEGER`, and `BYTE_INTEGER` in the package `STANDARD`. (See tests B86001C and B86001D.)

- . Based literals

An implementation is allowed to reject a based literal with a value exceeding `SYSTEM.MAX_INT` during compilation, or it may raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` during execution. This implementation raises `CONSTRAINT_ERROR` during execution. (See test E24101A.)

- . Expression evaluation.

Apparently some default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses no extra bits for extra range. (See test C35903A.)

CONFIGURATION INFORMATION

Apparently `CONSTRAINT_ERROR` is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Sometimes `CONSTRAINT_ERROR` is raised when a literal operand in a fixed point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is gradual. (See tests C45524A..Z.)

. Rounding.

The method used for rounding to integer is apparently round to even (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round to even. (see tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

. Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`. For this implementation:

Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises `NUMERIC_ERROR` (See test C36003A.)

`NUMERIC_ERROR` is raised when an array type with `INTEGER'LAST + 2` components is declared. (See test C36202A.)

`NUMERIC_ERROR` is raised when an array type with `SYSTEM.MAX_INT + 2` components is declared. (See test C36202B.)

A packed `BOOLEAN` array having a `'LENGTH` exceeding `INTEGER'LAST` raises `NUMERIC_ERROR` when the array type is declared (See test C52103X.)

A packed two-dimensional `BOOLEAN` array with more than `INTEGER'LAST` components raises `NUMERIC_ERROR` when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than `INTEGER'LAST` may raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR`

CONFIGURATION INFORMATION

either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises `NUMERIC_ERROR` when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before `CONSTRAINT_ERROR` is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, index subtype checks appear to be made as choices are evaluated (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

Not all choices are evaluated before `CONSTRAINT_ERROR` is raised if a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

- Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause

CONFIGURATION INFORMATION

is not supported, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are not supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are not supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are not supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are not supported. (See tests A39005E and C87B62C.)

Record representation clauses are supported, though there are some restrictions as given in Appendix B Item 4. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are not supported. (See test C87B62A.)

. Pragmas.

The pragma `INLINE` is not supported for procedures. The pragma `INLINE` is not supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

. Input/output.

The package `SEQUENTIAL_IO` cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package `DIRECT_IO` cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H,

CONFIGURATION INFORMATION

EE2401D, and EE2401G.)

Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO.
(See tests CE2102D and CE2102E.)

Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for
DIRECT_IO. (See tests CE2102F, CE2102I, and CE2102J.)

RESET and DELETE are supported for SEQUENTIAL_IO and
DIRECT_IO. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are supported for
SEQUENTIAL_IO and DIRECT_IO. (See tests CE2106A and
CE2106B.)

Overwriting to a sequential file does not truncate the file.
(See test CE2208B.)

An existing text file can be opened in OUT_FILE mode, can be
created in OUT_FILE mode, and can be created in IN_FILE
mode. (See test EE3102C.)

More than one internal file can be associated with each
external file for text I/O for both reading and writing.
(See tests CE2110B, CE2111D, CE3111A..E (5 tests), CE3114B,
and CE3115A.)

More than one internal file can be associated with each
external file for sequential I/O for both reading and
writing. (See tests CE2107A..D (4 tests) and CE2111D.)

More than one internal file can be associated with each
external file for direct I/O for both reading and writing
(See tests CE2107E..I (5 tests) and CE2111H.)

An external file associated with more than one internal
file can be deleted for SEQUENTIAL_IO, DIRECT_IO, and
TEXT_IO. (See test CE2110B).

Temporary sequential files are not given names. Temporary
direct files are not given names. (See tests CE2108A and
CE2108C.)

Generics.

Generic subprogram bodies can only be compiled in separate
compilations provided that no instantiations of the
corresponding generics occur prior to the compilation of the
generic body.

CONFIGURATION INFORMATION

Generic package bodies can only be compiled in separate compilations provided that no instantiations of the corresponding generics occur to the compilation of the generic body.

Generic subunits can only be compiled in separate compilations provided that no instantiations of the corresponding generic body (including all subunits), occur prior to the compilation of the generic body.

CHAPTER 3 TEST INFORMATION

3.1 TEST RESULTS

At the time of testing, version 1.9 of the ACVC comprised 3122 tests of which 25 had been withdrawn. Of the remaining tests, 399 were determined to be inapplicable to this implementation. Not all of the inapplicable tests were processed during testing; 327 executable tests that use floating-point precision exceeding that supported by the implementation were not processed. Modifications to the code, processing, or grading for 40 tests were required to successfully demonstrate the test objective. (See section 3.6)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						
	A	B	C	(C) 1481		2698	OTAL
Passed	103	1045	1474	17	8	44	2691
Failed	0	0	0	0	0	0	0
Inapplicable	7	6	381	0	10	2	406
Withdrawn	3	2	19	374		399	25
TOTAL	113	1053	1874	(C)			3122

TEST INFORMATION

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													
	2	3	4	5	6	7	8	9	10	11	12	13	14	TOTAL
Passed	175	449	460	245	166	98	139	326	129	36	232	3	240	2698
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	29	124	215	3	0	0	4	1	8	0	2	0	13	399
Withdrawn	2	13	2	0	0	1	2	0	0	0	2	1	2	25
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 25 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C	C35904A
C35A03E	C35A03R	C37213H	C37213J	C37215C	C37215E
C37215G	C37215H	C38102C	C41402A	C45614C	A74106C
C85018B	C87B04B	CC1311B	BC3105A	AD1A01A	CE2401H
CE3208A					

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 399 tests were inapplicable for the reasons indicated:

TEST INFORMATION

- . B23003D and B23003E are inapplicable because no restrictions are made on the maximum line length.
- . E28002A, E28002B, E28002D and E28005D use pragmas LIST and PAGE. These pragmas are not supported by this compiler.
- . C35A06N contains a fixed-point type declaration (line 17) for which this implementation chooses the base type to be the same as the type declared. As a consequence, a subsequent fixed-point type declaration (lines 31 & 32) that uses the earlier type's SAFE_LARGE value (+/-) as the bounds of its range, results in a type with 0 as its sole model number. This, in turn, causes CONSTRAINT_ERROR to be raised when the SAFE_SMALL and SAFE_LARGE values of the first type are passed as parameters of the later type. The AVO ruled that C35A06N was NA due to this unexpected result.
- . C35702A and C35702B use SHORT_FLOAT and LONG_FLOAT respectively, which are not supported by this implementation.
- . A39005B and C87B62A use length clauses with SIZE specifications for derived integer types or for enumeration types which are not supported by this compiler (See Appendix B Item 4).
- . A39005C..D (2 tests), C87B62B and C87B62D use length clauses with STORAGE_SIZE specifications for access types or for task types which are not supported by this implementation.
- . A39005E and C87B62C use length clauses with SMALL specifications which are not supported by this implementation.
- . A39005G uses a record representation clause that is not supported by this compiler (See Appendix B Item 4).
- . The following tests use LONG_INTEGER, which is not supported by this compiler.

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45631C	
C45632C	B52004D	C55B07A	B55B09C	

- . C45531I, C45531J, C45532I, and C45532J use fine 32 bit fixed point base types which are not supported by this compiler.
- . C45531K, C45531L, C45532K, and C45532L use coarse 32 bit fixed point base types which are not supported by this compiler.
- . C45531M, C45531N, C45532M, and C45532N use fine 48 bit fixed point base types which are not supported by this compiler.
- . C45531O, C45531P, C45532O, and C45532P use coarse 48 bit fixed point base types which are not supported by this compiler.

TEST INFORMATION

- . C4A013B uses a static value that is outside the range of the most accurate floating point base type. The declaration was rejected at compile time.
- . C96005B requires the range of type DURATION to be different from those of its base type; in this implementation they are the same.
- . CA2009F, CA2009C, BC3204C and BC3205D compile generic subprogram declarations or package specifications and their corresponding bodies in separate compilations. This compiler requires that no instantiations of the corresponding generics occur prior to the compilations of the generic body.
- . CA3004E, EA3004C, and LA3004A use the INLINE pragma for procedures, which is not supported by this compiler.
- . CA3004F, EA3004D and LA3004B use the INLINE pragma for functions, which is not supported by this compiler.
- . AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.
- . AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.
- . CE2107C, CE2107D, CE2107H, CE2107I, CE2108A, CE2108C and CE3112A are inapplicable because temporary files do not have names.
- . The following 327 tests require a floating-point accuracy that exceeds the maximum of 6 digits supported by this implementation:

C24113C..Y (23 tests)	C35705C..Y (23 tests)
C35706C..Y (23 tests)	C35707C..Y (23 tests)
C35708C..Y (23 tests)	C35802C..Z (24 tests)
C45241C..Y (23 tests)	C45321C..Y (23 tests)
C45421C..Y (23 tests)	C45521C..Z (24 tests)
C45524C..Z (24 tests)	C45621C..Z (24 tests)
C45641C..Y (23 tests)	C46012C..Z (24 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behaviour. Modification are made with the approval of the AVO, and are made in cases where legitimate implementation behaviour prevents the successful completion of an (otherwise) applicable test.

TEST INFORMATION

Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into sub-tests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behaviour that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 40 Class B tests.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A	B32103A	B43201A	B51001A
B23004A	B33102A	B44001A	B51003A
B28003A	B35103B	B44004A	B54A01C
B29001A	B37301B	B49007A	B54A20A
B2A003A	B38001C		B55A01A
B2A007A			
B2A010A			

B66001C	B71001C	B91001H	BA1101B
B91002F	BA1101C	B95001A	B95004B
B95061G	B95077A	B97101A	B97102A
B97102H			

BC1001A	BC1016A	BC1202E	BC2001D
BC2001E	BC3204B		

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the University of York Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behaviour on all inapplicable tests.

3.7.2 Test Method

Testing of the University of York Ada Compiler using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a Sun 3/50 operating under UNIX (SunOS 3.2)

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests

TEST INFORMATION

that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape.

The contents of the magnetic tape were not loaded directly onto the host computer.

The contents of the magnetic tape were loaded to a Sun 3/280 fileserver. Files, when required, were transferred via Ethernet from the fileserver to a Sun 3/50 Workstation.

After the test files were loaded to disk, the full set of tests was compiled on the Sun 3/50 and all executable tests were linked and run. Results were printed from the host computer.

The compiler was tested using command scripts provided by University of York and reviewed by the validation team. The compiler was tested using all default options settings.

Tests were compiled, linked, and executed (as appropriate) using 3 host computers. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

The validation team arrived at University of York on 25th November 1987, and departed after testing was completed on 26th November 1987.

APPENDIX A

CONFORMANCE STATEMENT

University of York has submitted the following conformance statement concerning the University of York Ada Compiler.

CONFORMANCE STATEMENT

DECLARATION OF CONFORMANCE

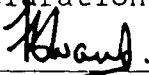
Compiler Implementor: University of York
Ada Validation Facility: National Computing Centre Limited UK
Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name: University of York Ada Compiler
Version : Release 3
Host Architecture : Sun 3/50 OS&VER : Unix (SunOS 3.2)
Target Architecture : same as host OS&VER : same as host

Implementor's Declaration

I, the undersigned, representing University of York, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that University of York is the owner of record of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.



University of York
Professor I C Wand
Head of Department (Computer Science)

Date: 2 December 1987.

*Ada is a registered trademark of the United States Government
(Ada Joint Program Office)

CONFORMANCE STATEMENT

Owner's Declaration

I, the undersigned, representing University of York, take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language ANSI/MIL-STD-1815A.

I C Wand

Date: 2 December 1987.

University of York
Professor I C Wand
Head of Department (Computer Science)

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the University of York Ada Compiler, Release 3 are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

1. Implementation-dependent pragma LIBNAME

This pragma is used to associate a subprogram body written in a language other than Ada with a corresponding Ada subprogram specification in a package specification. The other-language body is supplied in lieu of an Ada body. The pragma must be placed immediately before the specification of the subprogram for which the foreign body is to be supplied. The single argument to pragma LIBNAME is the link-symbol of the foreign body. For C this is the routine name (case is significant). This pragma may also apply to objects.

e.g.

```
-- Ada
package TEST is
    pragma LIBNAME("iadd");
    procedure IADD(I,J: in out INTEGER);
end TEST;
```

```
-- C body for IADD
void iadd(a,b)
int *a,*b;
{
    *a = *a + *b;
}
```

2. Implementation-dependent attributes

None.

3. Package SYSTEM

```
package SYSTEM is
    type ADDRESS is private;
    type NAME is (UNIX);
    SYSTEM_NAME : constant := UNIX;
    STORAGE_UNIT : constant := 8;
    MEMORY_SIZE : constant := 2147483647;
    MIN_INT : constant := -2147483648;
    MAX_INT : constant := 2147483647;
    MAX_DIGITS : constant := 6;
    MAX_MANTISSA : constant := 30;
    FINE_DELTA : constant := 0.000000001;
    TICK : constant := 0.01;
    subtype PRIORITY is INTEGER range 0..4;
end SYSTEM;
```

4. Representation clauses

Length clauses and interrupts are not implemented. There are some restrictions on record representation clauses as follows:-

- (1) The static simple expression in any alignment clause must be 1, 2 or 4
- (2) Component clauses for fields whose types are non-static or whose types depend upon discriminants are not allowed
- (3) Fields with a type which is not an array or record type must have a bit length which is less than or equal to 32
- (4) Fields with a type which is an array or record type must begin on a storage unit boundary (1 byte or 8 bits) and must have a field length which is a multiple of a storage unit
- (5) Fields with a discrete type may be packed to a field length which is not a multiple of a storage unit (as long as this is feasible for the range of values for the type) but the compiler will refuse to do this for array and record types.

5. Implementation-generated names

Not implemented.

6. Unchecked conversions

No restrictions.

7. Input-output packages

SEQUENTIAL_IO and DIRECT_IO may not be instantiated with unconstrained types.

TEXT_IO.ENumeration_IO will raise USE_ERROR if instantiated with numeric types.

Temporary files do not have names in this implementation.

Actual parameter to "FORM" must be the empty string("").

8. Main program

The main program should be a parameterless procedure otherwise the effect is undefined.

9. Generic bodies

Generic bodies must appear in the same file as their specifications or be already compiled when an instantiation is made.

10. Address Clauses

Address clauses are implemented, but not for interrupts.

11. Package Standard

Implementation Parameters

byte_integer'first	-128
byte_integer'last	127
byte_integer'size	8
short_integer'first	-32768
short_integer'last	32767
short_integer'size	16
integer'first	-2147483648
integer'last	2147483647
integer'size	32
float'size	32
float'first	-1.7014117331926443e38
float'last	1.7014117331926443e38
float'safe_large	1.7014117331926443e38
float'safe_small	2.9387362273803348e-39
float'safe_emax	127
float'digits	6
float'machine_emax	127
float'machine_emin	-128
float'machine_mantissa	24
float'machine_overflows	true
float'machine_rounds	true
float'machine_radix	2
duration'delta	0.0167
duration'first	-16777215.0
duration'last	16777215.0
system.address'size	32
system.system_name	unix
system.storage_unit	8
system.memory_size	2147483647
system.min_int	-2147483648
system.max_int	2147483647

APPENDIX F OF THE Ada STANDARD

system.max_digits	6
system.max_mantissa	30
system.fine_delta	0.000000001
system.tick	0.01
system.priority'first	0
system.priority'last	4
direct_io.count'last	2147483647
text_io.count'last	2147483647
field'last	40

For all fixed point types f :-

f'size	32
f'machine_overflows	true
f'machine_rounds	false
maximum input line length	no limit set by compiler

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

<u>Name and Meaning</u>	<u>Value</u>
\$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	1..127=>'A',128=>'1'
\$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	1..127=>'A',128=>'2'
\$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	1..63=>'A',64=>'3',65..128=>'A'
\$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	1..63=>'A',64=>'4',65..128=>'A'
\$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum length.	1..125=>'0',126..128=>'298'

TEST PARAMETERS

Name and Meaning	Value
\$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	1..122=>0,123..128=>'69.0E1'
\$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	"1..64=>'A'"
\$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	"1..63=>'A',64=>'1'"
\$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	1..108=>' '
\$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
\$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	40
\$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid character or is too long.	MUCH_TOO_LONG_NAME_FOR_A_FILE
\$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	MUCH_TOO_LONG_NAME_FOR_A_FILE
\$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	2147483647.0

TEST PARAMETERS

Name and Meaning	Value
\$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE' LAST.	2147483648.0
\$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	MUCH_TOO_LONG_NAME_FOR_A_FILE
\$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	MUCH_TOO_LONG_NAME_FOR_A_FILE
\$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483648
\$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
\$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2_147_483_648
\$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE' FIRST and DURATION'FIRST or any value in the range of DURATION.	-2147483647.0
\$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE' FIRST	-2147483648.0
\$MAX-DIGITS Maximum digits supported for floating-point types.	6
\$MAX_IN_LEN Maximum input line length permitted by the implementation.	128
\$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647

TEST PARAMETERS

Name and Meaning	Value
\$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2_147_483_648
\$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..2=>'2:',3..125=>'0',126..128=>'11:'
\$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..3=>'16:',4..124=>'0',125..128=>'F.E:'
\$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	"1..125=>'A',126=>'1'"
\$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
\$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	BYTE_INTEGER
\$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFFD#

APPENDIX D
WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 25 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- B28003A: A basic declaration (line 36) wrongly follows a later declaration.
- E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ALMP.
- C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.
- C35502P: The equality operators in lines 62 & 69 should be inequality operators.
- A35902C: Line 17's assignment of the nominal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR for that value lies outside of the actual range of the type.
- C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.
- C35A03E, These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.
- C35A03R These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

WITHDRAWN TESTS

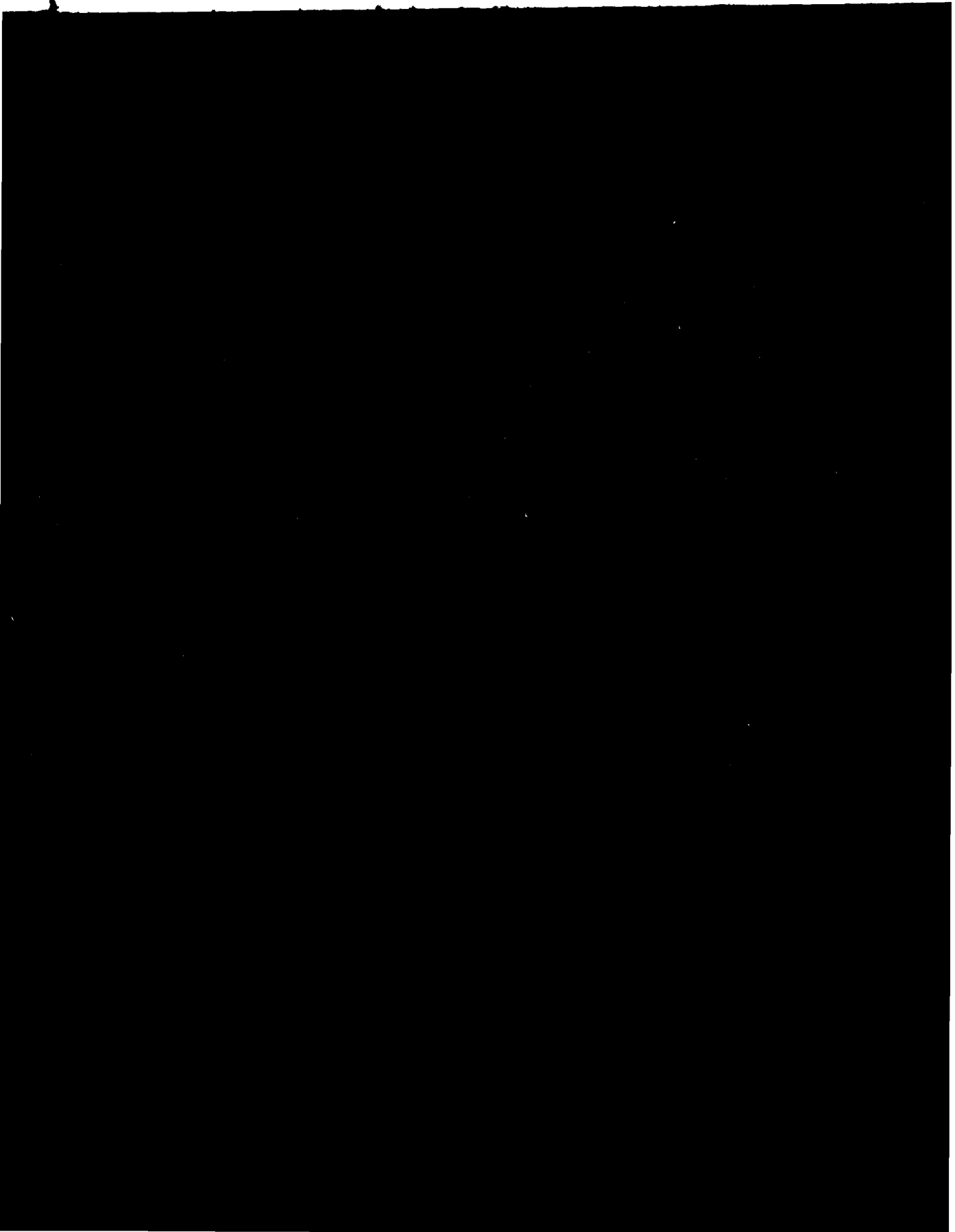
- C37213H The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.
- C37213J The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.
- C37215C Various discriminant constraints are wrongly expected to be incompatible with type CONS.
- C37215E Various discriminant constraints are wrongly expected to be incompatible with type CONS.
- C37215G Various discriminant constraints are wrongly expected to be incompatible with type CONS.
- C37215H Various discriminant constraints are wrongly expected to be incompatible with type CONS.
- C38102C The fixed-point conversion on line 25 wrongly raises CONSTRAINT_ERROR.
- C41402A 'STORAGE-SIZE' is wrongly applied to an object of an access type.
- C45614C REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).
- A74016C A bound specified in a fixed-point subtype declaration lies outside that calculated for the base type, raising CONSTRAINING_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48 and 252 & 253 of the four tests, respectively (and possibly elsewhere).
- C85018B A bound specified in a fixed-point subtype declaration lies outside that calculated for the base type, raising CONSTRAINING_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48 and 252 & 253 of the four tests, respectively (and possibly elsewhere).
- C87B04B A bound specified in a fixed-point subtype declaration lies outside that calculated for the base type, raising CONSTRAINING_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48 and 252 & 253 of the four tests, respectively (and possibly elsewhere).
- CC1311B A bound specified in a fixed-point subtype declaration lies outside that calculated for the base type, raising CONSTRAINING_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48 and 252 & 253 of the four tests, respectively (and possibly elsewhere).
- BC3105A Lines 159..168 are wrongly expected to be incorrect; they are correct.

WITHDRAWN TESTS

- AD1A01A The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.
- CE2401H The record aggregates in lines 105 & 117 contain the wrong values.
- CE3208A This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.

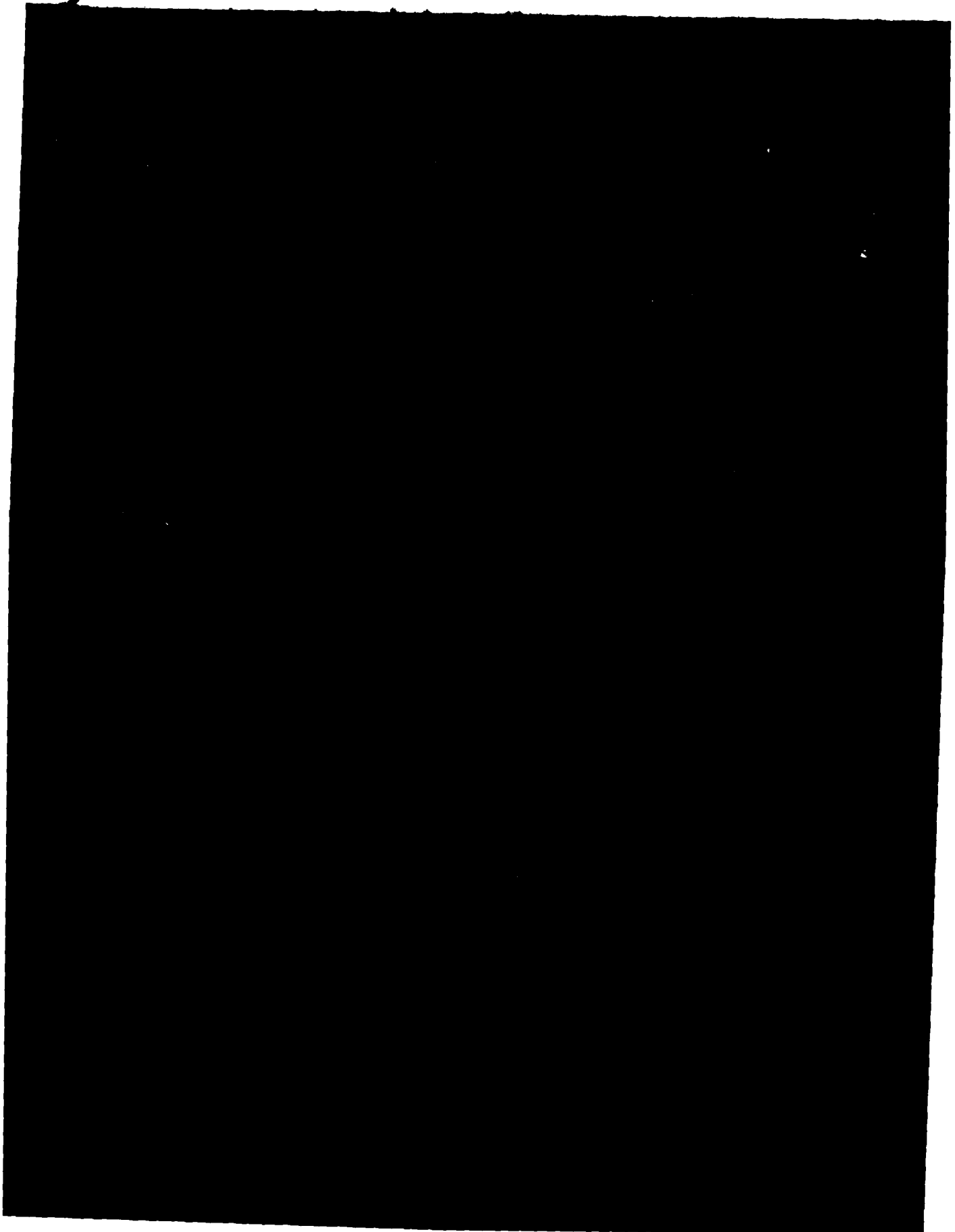
MED
-8





subtype PRIORITY is INTEGER range 0..4,
end SYSTEM;

Appendix B Page 2 of 5



The main program should be a parameterless procedure otherwise
the effect is undefined.